Privacy for Computations

Vicenç Torra

February 2024

Umeå University, Sweden

V. Torra (2022) A guide to data privacy, Springer (Chapter 5)

Outline

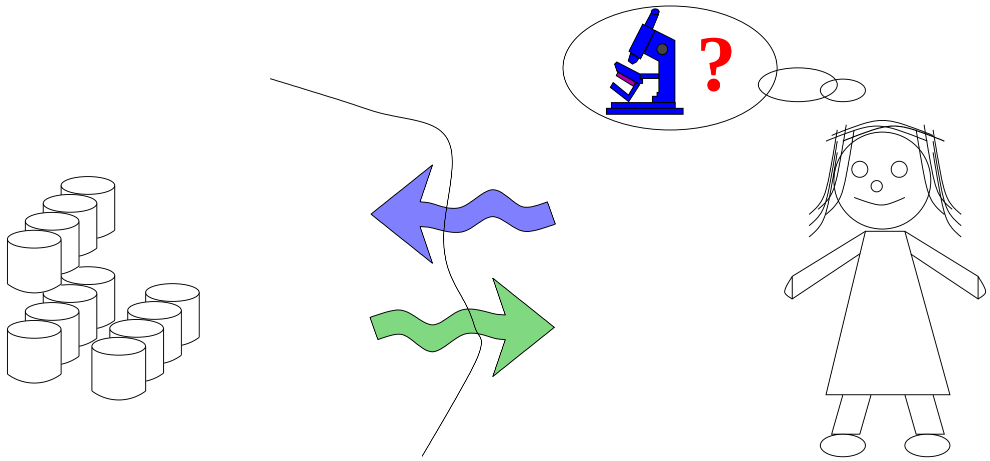
1. Computation-driven approaches
   - Differential privacy
   - Centralized approach: trusted third party
   - Distributed approach: secure multiparty computation

Privacy for computations

Introduction

Introduction

- The researcher computes a function without accessing the data

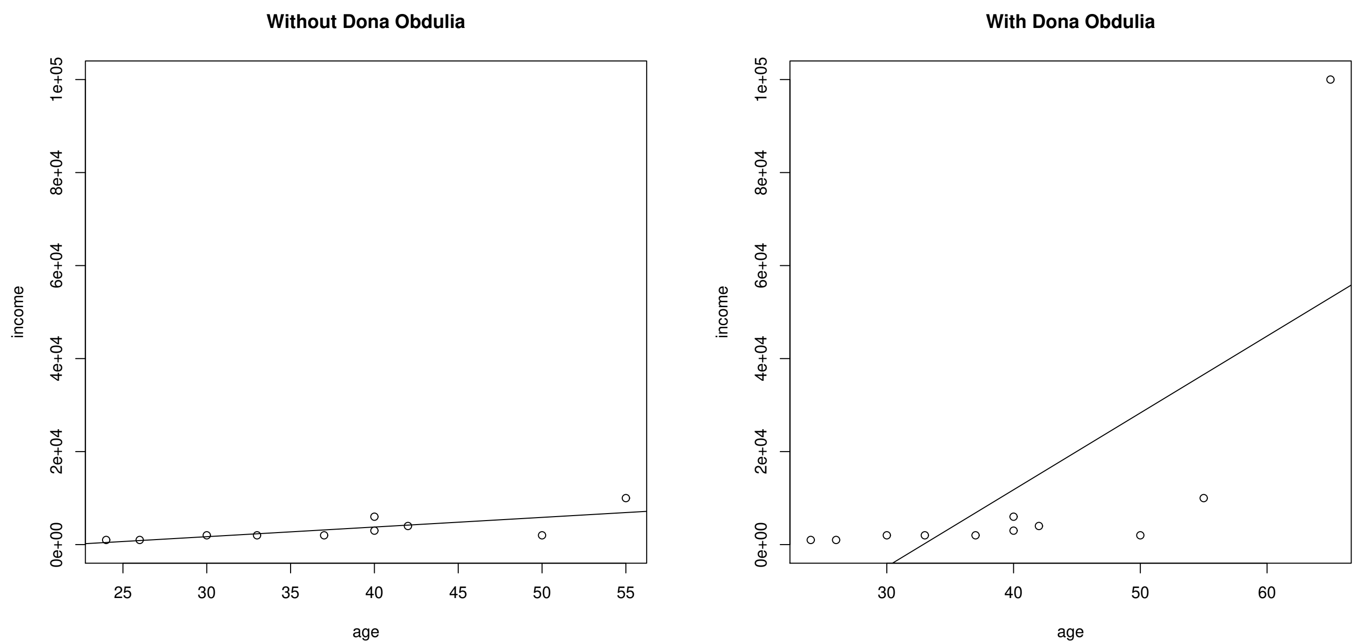


Data is sensitive: computation leads to disclosure

- Motivating example #1 (Case #2. Sharing a computation)
  - Q: Mean income of admitted to hospital unit (e.g., psychiatric unit) for a given Town (Bunyola)?
  - Mean income is not "personal data", is this ok ? NO!!:
  - Example 1000 2000 3000 2000 1000 6000 2000 10000 2000 4000 $\Rightarrow$ mean $= 3300$
  - Adding Ms. Rich's salary 100,000 Eur/month: mean $= 12090{,}90$!
    (a extremely high salary changes the mean significantly)
    $\Rightarrow$ We infer Ms. Rich from Town was attending the unit

Data is sensitive: computation leads to disclosure

- Motivating example #2 (Case #2. Sharing a computation)



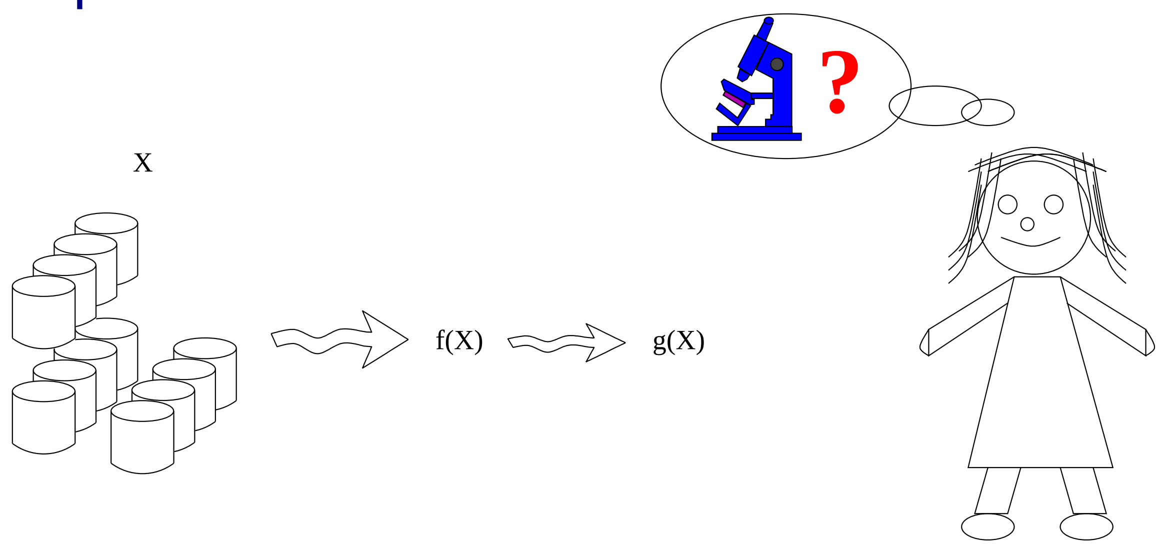
- Regression of income with respect to age with (right) and without (left) the record of Dona Obdúlia
  - income = -4524.2 + 207.5 age (without Ms. Rich = Dona Obdúlia)
  - income = -54307 + 1652 age (with Ms. Rich = Dona Obdúlia)

Privacy models (review)

Privacy for computations: privacy models I

Privacy models. Computing a function (centralized)

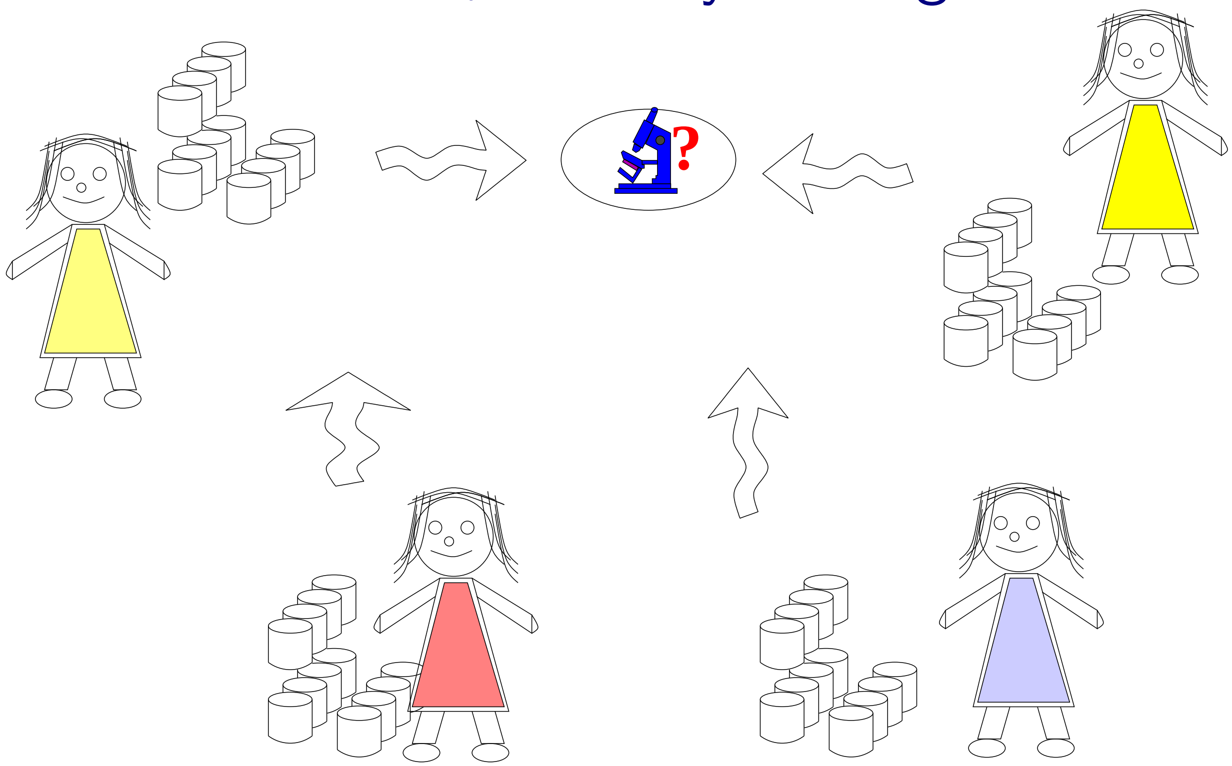
- **Differential privacy.** The output of a query to a database should not depend (much) on whether a record is in the database or not.
- **Integral privacy.** Inference on the databases. E.g., changes have been applied to a database.
- **Homomorphic encryption.** We want to avoid access to raw data and partial computations.



Privacy for computations: privacy models II

Privacy models. Computing a function (distributed)

- **Secure multiparty computation.** Several parties want to compute a function of their databases, but only sharing the result.



Introduction: Summary

- Important assumptions
  - We know the function to compute
  - Data is not shared, only the output of the function
  - Partial computations are not shared, only the output of the function
  - We do not want that the output of the function leads to disclosure

Introduction: Summary

5 Privacy for Computations, Functions, and Queries
5.1 Differential Privacy Mechanisms
5.1.1 Differential Privacy Mechanisms for Numerical Data
5.1.2 Composition Theorems
5.1.3 Differential Privacy Mechanisms for Categorical Data
5.1.4 Properties of Differential Privacy
5.1.5 Machine Learning
5.1.6 Concluding Remarks
5.2 Secure Multiparty Computation Protocols
5.2.1 Assumptions on Data and on Adversaries
5.2.2 Computing a Distributed Sum
5.2.3 Secure Multiparty Computation and Inferences
5.2.4 Computing the Exclusive OR Function
5.2.5 Secure Multiparty Computation for Other Functions
5.3 Bibliographical Notes

Secure Multiparty computation

Centralized approach: Trusted third party

Trusted third party

Computation-driven approaches/multiple databases: centralized

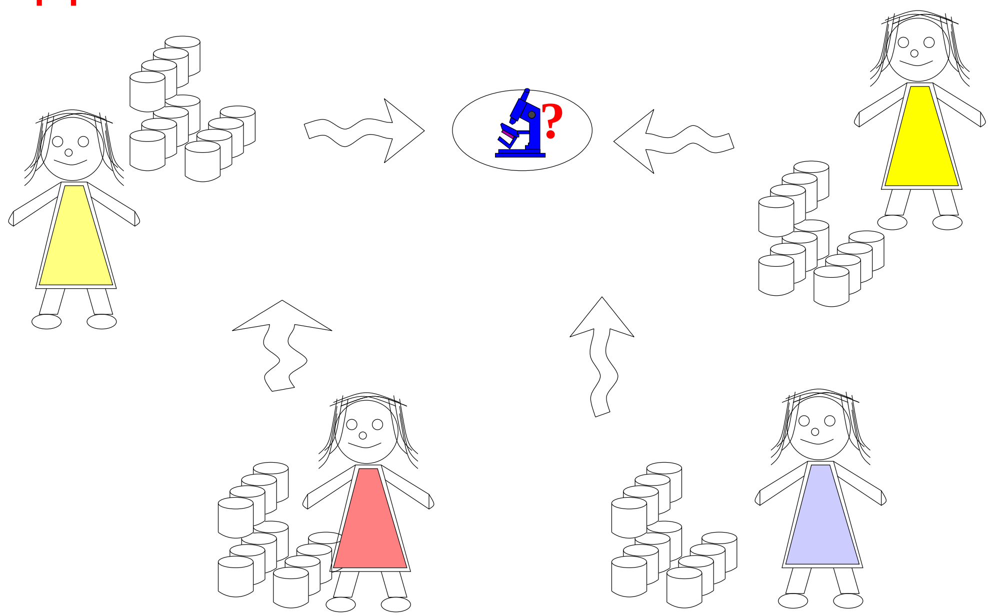
- **Example.** Parties $P_1, \ldots, P_n$ own databases $DB_1, \ldots, DB_n$. The parties want to compute a function, say f, of these databases (i.e., $f(DB_1, \ldots, DB_n)$) without revealing unnecessary information. In other words, after computing $f(DB_1, \ldots, DB_n)$ and delivering this result to all P_i, what P_i knows is nothing more than what can be deduced from his DB_i and the function f.
- So, the computation of f has not given P_i any extra knowledge.

Distributed approach: secure multiparty computation

Secure multiparty computation

Computation-driven approaches/multiple databases: distributed

- The centralized approach as a reference



Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Compute the sum of salaries of 4 people: Aine, Brianna, Cathleen, and Deirdre.
  We denote these salaries by s_1, s_2, s_3, and s_4, respectively.
- Each person's salary is confidential and they do not want to share.
- Define a protocol to compute involving only the 4 people (no trusted third party).
- Assume that the sum lies in the range $[0, n]$.

☐ Example with 4 people. Similar method applies with other number of people.

☐ We use public-key cryptography. I.e., each party requires two separate keys: a private and a public one. This is also known as asymmetric cryptography.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.
- Brianna decrypts Aine's message with Brianna's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 \ mod \ n$) to Cathleen encrypted with Cathleen's public key.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.
- Brianna decrypts Aine's message with Brianna's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 \ mod \ n$) to Cathleen encrypted with Cathleen's public key.
- Cathleen decrypts Brianna's message with Cathleen's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 \ mod \ n$) to Deirdre encrypted with Deirdre's public key.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.
- Brianna decrypts Aine's message with Brianna's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 \ mod \ n$) to Cathleen encrypted with Cathleen's public key.
- Cathleen decrypts Brianna's message with Cathleen's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 \ mod \ n$) to Deirdre encrypted with Deirdre's public key.
- Deirdre decrypts Cathleen's message with Deirdre's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 + s_4 \ mod \ n$) to Aine encrypted with Aine's public key.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.
- Brianna decrypts Aine's message with Brianna's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 \ mod \ n$) to Cathleen encrypted with Cathleen's public key.
- Cathleen decrypts Brianna's message with Cathleen's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 \ mod \ n$) to Deirdre encrypted with Deirdre's public key.
- Deirdre decrypts Cathleen's message with Deirdre's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 + s_4 \ mod \ n$) to Aine encrypted with Aine's public key.
- Aine decrypts Deirdre's message with Aine's private key. She substracts (modulo n) the random number r added in the first step, obtaining in this way $s_1 + s_2 + s_3 + s_4$ (this will be in $[0, n]$).

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Aine adds a secret random number, say r (uniformly chosen in $[0, n]$) to her salary and sends it to Brianna encrypted with Brianna public key. Addition is modulo n. In this way, the outcome of $r + s_1 \ mod \ n$ will be a number uniformly distributed in $[0, n]$ and so Brianna will learn nothing about the actual value of s_1.
- Brianna decrypts Aine's message with Brianna's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 \ mod \ n$) to Cathleen encrypted with Cathleen's public key.
- Cathleen decrypts Brianna's message with Cathleen's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 \ mod \ n$) to Deirdre encrypted with Deirdre's public key.
- Deirdre decrypts Cathleen's message with Deirdre's private key, adds her salary (modulo n) and sends the result (i.e., $r + s_1 + s_2 + s_3 + s_4 \ mod \ n$) to Aine encrypted with Aine's public key.
- Aine decrypts Deirdre's message with Aine's private key. She substracts (modulo n) the random number r added in the first step, obtaining in this way $s_1 + s_2 + s_3 + s_4$ (this will be in $[0, n]$).
- Aine announces the result to the participants.

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- This protocol assumes that all of the participants are honest
- A participant can lie about her salary.
- Aine can announce a wrong addition.
- Participants can collude. E.g.,
  - Brianna and Deirdree can share their figures to find the salary of Cathleen

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- Solving collusion.
  - Each salary is divided into shares.
  - The sum of each share is computed individually.
  - Different paths are used for different shares in a way that neighbors are different.

    To compute any s_i all neighbors of all paths are required.
  - Different number of shares imply different minimum coalition sizes for violating security

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

Important observation

- This method is compliant with the privacy model selected: Secure multiparty computation
- This method is not compliant with other privacy models: differential privacy

We can define appropriate methods that satisfy multiple privacy models

- E.g., method that computes differentially private secure sum

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed.

- Yao's millionaire problem. Alice and Bob want to know who is richer, but they do not want to tell the other how much money they have. This is the secure computation of $a > b$.
- Secure set union.
- Scalar product. Alice with vector x and Bob with vector y want to compute xy.

Secure multiparty computation

Computation-driven approaches/multiple databases: distributed.

- Machine learning and data mining methods.
- Parties can be seen as sharing the schema of a database.
- Two types of problems usually considered.
  - Vertically partitioned data. Parties (data holders) have information on the same individuals but different attributes.
  - Horizontally partitioned data. Parties (data holders) have information on different individuals but on the same attributes (i.e., the share the database schema).

Secure multiparty computation

Computation-driven approaches/multiple databases: distributed

Privacy leakage for the distributed approach is usually analyzed considering two types of adversaries.

Secure multiparty computation

Computation-driven approaches/multiple databases: distributed

Privacy leakage for the distributed approach is usually analyzed considering two types of adversaries.

- **Semi-honest adversaries.** Data owners follow the cryptographic protocol but they analyse all the information they get during its execution to discover as much information as they can.
- **Malicious adversaries.** Data owners try to fool the protocol (e.g. aborting it or sending incorrect messages on purpose) so that they can infer confidential information.

Computing the Exclusive OR Function

Exclusive OR

Dining Cryptographer network. DC-net (Chaum, 1985)

Exclusive OR

Dining Cryptographer network. DC-net (Chaum, 1985)

- Sender anonymity, or a secure multi-party computation of the function OR.

Exclusive OR

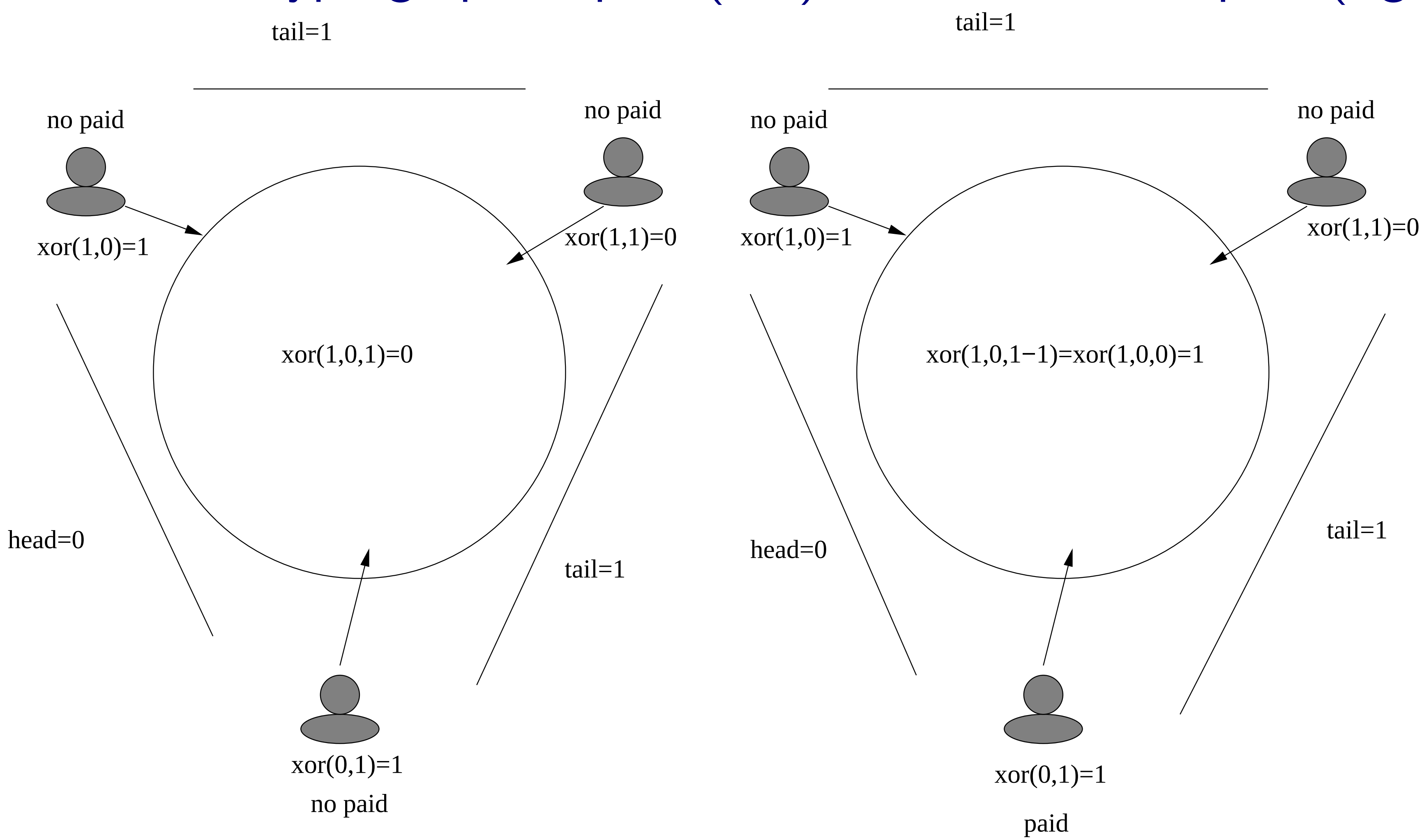
Dining Cryptographer network. DC-net (Chaum, 1985)

- Sender anonymity, or a secure multi-party computation of the function OR.
- **Problem.** Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maître d'hôtel for the bill to be paid anonymously. One of the cryptographers might be paying the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers repect each other's right to make an anonymous payment, but they wonder if NSA is paying.

Exclusive OR

Dining Cryptographer network.

- Graphical representation of the solution
  (None of the cryptographers paid (left) and one of them paid (right))



Exclusive OR

Dining Cryptographer network. Steps of the process (I)

Step 1. Each cryptographer flips a coin and shares its outcome with the crytographer on the right. Let us represent tails and heads by 1 and 0, respectively. Let $coin_i$ be the outcome of the coin of the ith cryptographer.

Exclusive OR

Dining Cryptographer network. Steps of the process (I)

Step 1. Each cryptographer flips a coin and shares its outcome with the crytographer on the right. Let us represent tails and heads by 1 and 0, respectively. Let $coin_i$ be the outcome of the coin of the ith cryptographer.

Step 2. All cryptographers find whether the two coins they know about (the one they flipped and the one their left-hand neighbor flipped) fell on the same side or not. Let us use the xor on the results of the two coins to represent the computation of the cryptographer: $c_i = xor(coin_i, coin_{(i-1) \mod 3})$.

Exclusive OR

Dining Cryptographer network. Steps of the process (II)

Step 3. If a cryptographer is the payer, then the answer is the opposite of what is observed. Otherwise, says what is observed. Formally, let us represent the statement of the ith cryptographer by c'_i, then

$$c'_i = \begin{cases} c_i & \text{if the } i\text{th cryptographer did not pay the meal} \\ 1 - c_i & \text{if the } i\text{th cryptographer paid the meal.} \end{cases} \tag{1}$$

Exclusive OR

Dining Cryptographer network. Steps of the process (II)

Step 3. If a cryptographer is the payer, then the answer is the opposite of what is observed. Otherwise, says what is observed. Formally, let us represent the statement of the ith cryptographer by c'_i, then

$$c'_i = \begin{cases} c_i & \text{if the } i\text{th cryptographer did not pay the meal} \\ 1 - c_i & \text{if the } i\text{th cryptographer paid the meal.} \end{cases} \quad (1)$$

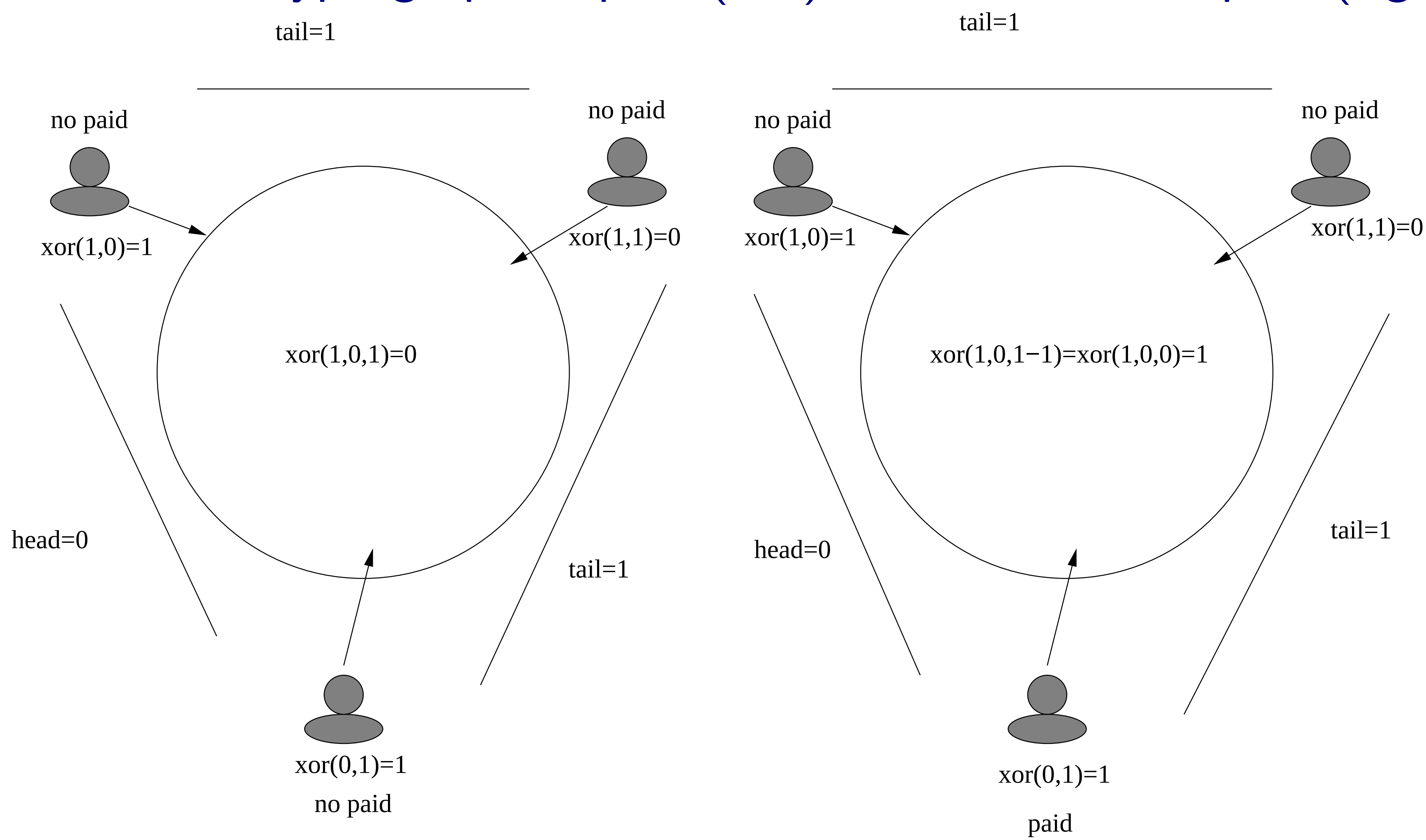
Step 4. Then, let s be the sum of the values c'_i. If the sum is even, no one paid. If odd, one crytographer paid. The xor function can be used for this purpose.

$$xor(c'_1, c'_2, c'_3)$$

Exclusive OR

Dining Cryptographer network.

- Graphical representation of the solution
  (None of the cryptographers paid (left) and one of them paid (right))



Exclusive OR

Dining Cryptographer network. Properties (I)

- This protocol can be generalized to an arbitrary number of crytographers.

Exclusive OR

Dining Cryptographer network. Properties (I)

- This protocol can be generalized to an arbitrary number of crytographers.
  - **Protocol.** Each crytographer needs a secret bit with each other participant. Each cryptographer computes the sum modulo two (or the xor function of all the bits). Then, the ith cryptographer applies the function above to determine c_i' from c_i (as above). Then, as in Step 4 above, let s be the sum of the values c_i'. If the sum is even, no one paid. If odd, one crytographer paid.

Exclusive OR

Dining Cryptographer network. Properties (II)

- Main problems:
  - (i) malicious participants make the output useless;
  - (ii) for n participants we need n^2 communications (one for each pair of participants).
- Only one participant can transmit a bit at a time. Two bits from different participants would cancel each other and would not be detected.

Exclusive OR: Unobservability

Unobservability. Undetectability and anonymity against other subjects

- Dining cryptographer networks

Secure multiparty computation

Computation-driven approaches/multiple databases/distributed. Sum

- We can also apply Shamir's secret sharing approach to this problem

Homomorphic encryption

Homomorphic encryption

- Procedure
  - Encrypt the data
  - Operate/compute over the encrypted data (no access to the secret key)
  - Result is encrypted

Homomorphic encryption

- Homomorphism: map that preserves the operations of the structures
  - A, B: two algebraic structures of the same type
  - $f : A \to B$ a map between A and B
  - operations $*$, $\circ$ on A and B
    $(A, *)$, $(B, \circ)$
  - $f(a * b) = f(a) \circ f(b)$

- Example:
  - $(\mathbb{R}, +)$
  - $(\mathbb{R}, \cdot)$
  - $f(a) = e^a$
  - Because, $e^{a+b} = e^a \cdot e^b$

Homomorphic encryption

- Types of homomorphic encryption
  - Partially homomorphic encryption: one type of gate (i.e., one operation), e.g., addition or multiplication.
  - Somewhat homomorphic encryption schemes: two types of gates (e.g., addition and multiplication), but only for a subset of circuits (not all composition of gates are possible).
  - Fully homomorphic encryption (FHE) allows the evaluation of arbitrary circuits composed of multiple types of gates of unbounded depth and is the strongest notion of homomorphic encryption.

Homomorphic encryption

- Partially homomorphic encryption:
  - ElGamal cryptosystem: modular multiplications
  - Paillier cryptosystem: additive homomorphic cryptosystem
- Fully homomorphic encryption:
  - Craig Gentry, Amit Sahai, and Brent Waters (GSW)

The case of federated Learning

Federated learning

- Motivation
  - Symbolic models (decision trees) vs. numerical models (deep learning)
  - Comparison of different privacy models: local and global
  - Who we trust? (privacy as a matter of trust)

Federated learning

- Privacy in federated learning and trust
  - Local privacy: The agent does not trust the system
    Local-DP / k-anonymity / privacy for re-identification

Federated learning

- Privacy in federated learning and trust
  - Local privacy: The agent does not trust the system
    Local-DP / k-anonymity / privacy for re-identification
  - Global privacy: Only globally we can really protect individuals
    (global) DP

Federated learning

- Privacy in federated learning and trust
  - Local privacy: The agent does not trust the system
    Local-DP / k-anonymity / privacy for re-identification
  - Global privacy: Only globally we can really protect individuals
    (global) DP
  - Infrastructure: No one trusts any one
    Secure multiparty computation

Federated learning

- Privacy in federated learning and trust
  - Local privacy: The agent does not trust the system
    Local-DP / k-anonymity / privacy for re-identification
  - Global privacy: Only globally we can really protect individuals
    (global) DP
  - Infrastructure: No one trusts any one
    Secure multiparty computation
  - Data in the cloud
    Homomorphic encryption

Federated learning

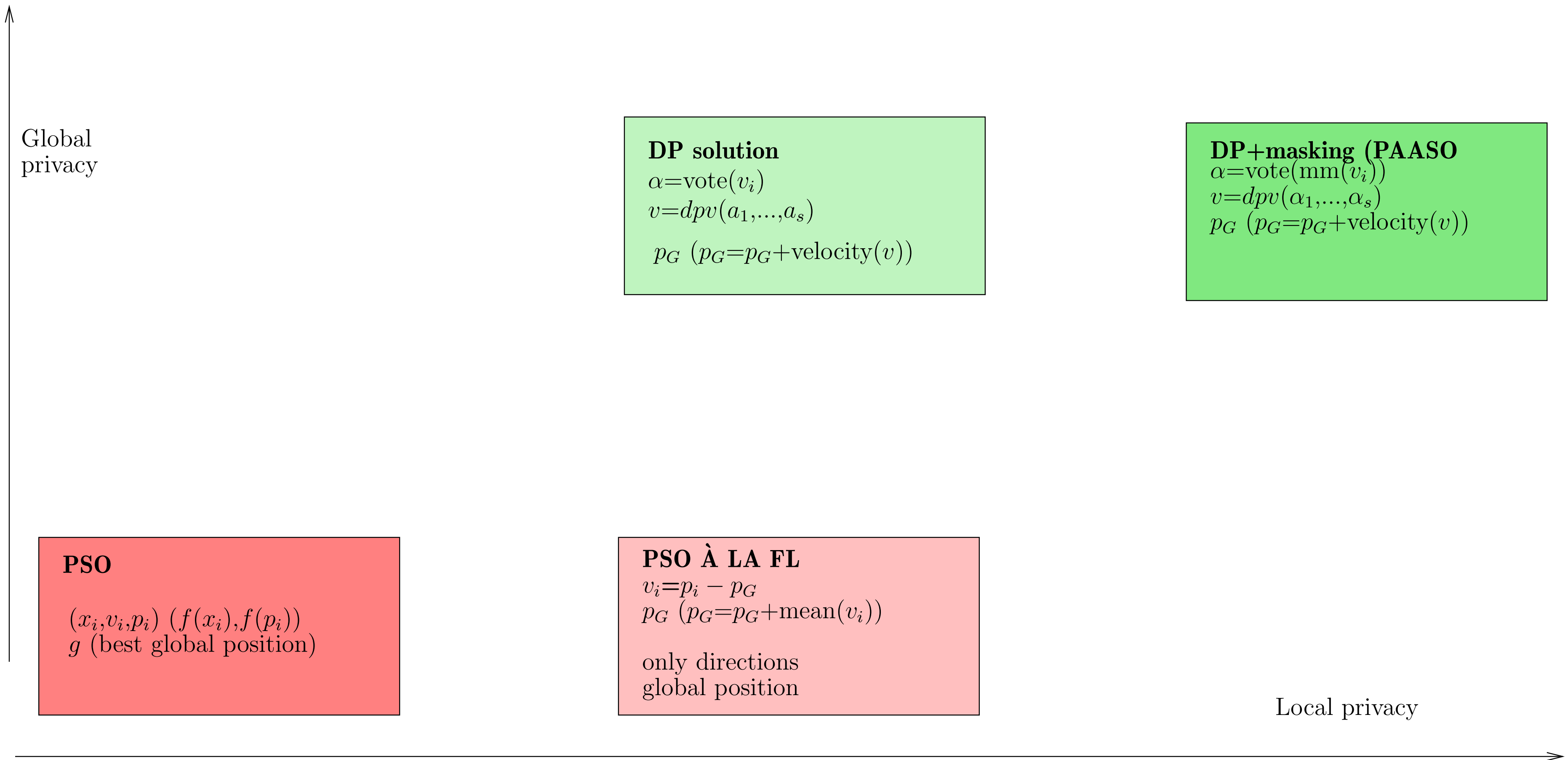
- Framework based on Particle Swarm Optimization (PSO)
  - Function to minimize
  - A set of particles (moving, position, direction) to find optimal
  - Privacy as a matter of trust

- Privacy at different levels
  - Symbolic vs. numerical PSO (less precision/voting/masking)
    - ▷ PSO is numerical, public (no-privacy): directions and positions
    - ▷ PSO à la FL (i.e., privacy): discrete directions (set of possible angles) – voting
  - Local vs. global privacy (individuals/clients vs. server)
    - ▷ Local: Masking vote before casting it (PRAM)
    - ▷ Global: Differentially private voting using DP-Random dictatorship[1]

[1] V. Torra, Random dictatorship for privacy-preserving social choice, Int. J. of Inf Sec, 19:5 (2020)

Federated learning

- PSO + FL = PAASO: Privacy-aware agent swarm optimization



Federated learning

- General comments PAASO with 2D problems[2]
  - In general, privacy mechanisms do not avoid convergence. It is slower. (this can be of concern, of course, rounds=information)
  - In terms of convergence, PSO and FL are best.
  - Local protection (PRAM) does not have strong effect.

- On the parameters
  - Number of options in voting, low effect
  - Number of agents, key factor (50, 100, and 200)
  - Particular parameters depend on the problem + privacy strategy

[2] V Torra et al., PSO + FL = PAASO: particle swarm optimization + federated learning = privacy-aware agent swarm optimization. Int. J. Inf. Sec. (2022)

Federated learning

- An example:
  - Mean objective function for 20 executions for FL, aDRD, and bDRD. Function f_4, number of voting alternatives $k_\alpha = 8$, 50 agents, $\phi_p = \phi_g = 2.00$. $p_c = 1.0$.
  - (left) $\omega = 4.00$, $\omega_G = 0.005$; (right) $\omega = 0.005$, $\omega_G = 0.01$
  - Generalized Rosenbrock's function ($x_1, x_2 \in [-2.0, 2.0]$):

$$f_4(x_1, x_2) = 100 * (x_2 - x_1 * x_1)^2 + (x_1 - 1)^2$$